Deep Counterfactual Regret Minimization

Noam Brown*
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
noamb@cs.cmu.edu

Adam Lerer*
Facebook AI Research
New York, NY 10003
alerer@fb.com

Sam Gross
Facebook AI Research
New York, NY 10003
sgross@fb.com

Tuomas Sandholm
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
sandholm@cs.cmu.edu

Abstract

Counterfactual Regret Minimization (CFR) is the leading algorithm for solving large imperfect-information games. It iteratively traverses the game tree in order to converge to a Nash equilibrium. In order to deal with extremely large games, CFR typically uses domain-specific heuristics to simplify the target game in a process known as *abstraction*. This simplified game is solved with tabular CFR, and its solution is mapped back to the full game. This paper introduces *Deep Counterfactual Regret Minimization (Deep CFR)*, a form of CFR that obviates the need for abstraction by instead using deep neural networks to approximate the behavior of CFR in the full game. We show that Deep CFR is principled and achieves strong performance in the benchmark game of heads-up no-limit Texas hold'em poker. This is the first successful use of function approximation in CFR for large games.

1 Introduction

Imperfect-information games model strategic interactions between multiple agents with only partial information. They are widely applicable to real-world domains such as negotiations, auctions, and cybersecurity interactions. Typically in such games, one wishes to find an approximate equilibrium in which no player can improve by deviating from the equilibrium.

The most successful family of algorithms for imperfect-information games have been variants of *Counterfactual Regret Minimization (CFR)* [29]. CFR is an iterative algorithm that converges to a Nash equilibrium in two-player zero-sum games. In order to deal with extremely large imperfect-information games, *abstraction* is typically used to simplify a game by bucketing similar states together and treating them identically. The simplified (abstracted) game is approximately solved via tabular CFR. Forms of tabular CFR have been used in all recent milestones in the benchmark domain of poker [2, 21, 4]. However, constructing an effective abstraction requires extensive domain knowledge and the abstract solution may only be a coarse approximation of a true equilibrium.

In constrast, reinforcement learning has been successfully extended to large state spaces by using function approximation with deep neural networks rather than a tabular representation of the policy (deep RL). This approach has led to a number of recent breakthroughs in constructing strategies

*Equal contribution

Preprint. Work in progress.

in large MDPs [20] as well as in zero-sum complete-information games such as Go [25, 24][2]. Importantly, deep RL can learn good strategies with relatively little domain knowledge for the specific game [25].

However, most popular RL algorithms do not converge to good policies (equilibria) in imperfect-information games in theory or in practice.

Rather than use tabular CFR with abstraction, this paper introduces a form of CFR, which we refer to as Deep Counterfactual Regret Minimization, that uses function approximation with deep neural networks to approximate the behavior of tabular CFR on the full, unabstracted game.

2 Notation and Background

In an imperfect-information extensive-form (that is, tree-form) game there is a finite set of players, $\mathcal{P}$. A *node* (or history) h is defined by all information of the current situation, including private knowledge known to only one player. $A(h)$ is the actions available in a node and $P(h)$ is the unique player who acts at that node. If action $a \in A(h)$ leads from h to h', then we write $h \cdot a = h'$. H is the set of all node in the game tree. $Z \subseteq H$ are terminal nodes for which no actions are available. For each player $p \in \mathcal{P}$, there is a payoff function $u_p : Z \to \mathbb{R}$. In this paper we assume $P = \{1, 2\}$ and $u_1 = -u_2$ (the game is two-player zero-sum). We denote the range of payoffs in the game by Δ.

Imperfect information is represented by *information sets* (infosets) for each player $p \in \mathcal{P}$. For any infoset I belonging to p, all nodes $h, h' \in I$ are indistinguishable to p. Moreover, every non-terminal node $h \in H$ belongs to exactly one infoset for each p. We represent the set of all infosets belonging to p where p acts by $\mathcal{I}_p$.

A strategy (or policy) $\sigma(I)$ is a probability vector over actions for acting player p in infoset I. The probability of a particular action a is denoted by $\sigma(I, a)$. Since all states in an infoset belonging to p are indistinguishable, the strategies in each of them must be identical. We define σ_p to be a strategy for p in every infoset in the game where p acts. A strategy profile σ is a tuple of strategies, one for each player. The strategy of every player other than p is represented as σ_{-p}. $u_p(\sigma_p, \sigma_{-p})$ is the expected payoff for p if all players play according to the strategy profile $\langle \sigma_p, \sigma_{-p} \rangle$.

$\pi^\sigma(h) = \Pi_{h' \to a \sqsubseteq h} \sigma_{P(h)}(h', a)$ is the probability h is reached if all players play according to σ. $\pi^\sigma_p(h)$ is the contribution of p to this probability (that is, the probability of reaching h if chance and all players other than p always choose actions leading to h, but p plays according to σ_p). $\pi^\sigma_{-p}(h)$ is the contribution of chance and all players other than p (that is, the probability of reaching h if p chooses actions leading toward h, but chance and all players other than p play according to σ_{-p}). $\pi^\sigma_{-p}(I) = \sum_{h \in I} \pi^\sigma_{-p}(h)$ is the probability of reaching I if p chooses actions leading toward I, but chance and all players other than p play according to σ_{-p}.

A *best response* to σ_p is a strategy $BR(\sigma_p)$ such that $u_p\big(\sigma_p, BR(\sigma_p)\big) = \max_{\sigma'_{-p}} u_i(\sigma_p, \sigma'_{-p})$. A *Nash equilibrium* σ^* is a strategy profile where everyone plays a best response: $\forall p$, $u_p(\sigma^*_p, \sigma^*_{-p}) = \max_{\sigma'_p} u_p(\sigma'_p, \sigma^*_{-p})$ [22]. The *exploitability* $e(\sigma_p)$ of a strategy σ_p in a two-player zero-sum game is how much worse it does versus a best response compared to a Nash equilibrium strategy. Formally, $e(\sigma_p) = u_p\big(\sigma^*_p, BR(\sigma^*_p)\big) - u_p\big(\sigma_p, BR(\sigma_p)\big)$. We measure *total exploitability* $\sum_{p \in P} e(\sigma_p)$.

2.1 Counterfactual Regret Minimization (CFR)

CFR is an iterative algorithm that maintains a *regret* value (defined later) for each action in each infoset, and uses those values to define a strategy for each player on each iteration. Based on those strategies, CFR updates the regret values.

The expected value (or simply *value*) to p at node h given that all players play according to strategy profile σ from that point on is defined as $v^\sigma_p(h)$. The value to p at infoset I where p acts is the weighted average of the value of each node in the infoset, where the weight is proportional to p's belief that they are in that node conditional on knowing they are in I. Formally, $v^\sigma(I) = \sum_{h \in I} \left(\pi^\sigma_{-p}(h|I) v^\sigma_p(h)\right)$ and $v^\sigma(I, a) = \sum_{h \in I} \left(\pi^\sigma_{-p}(h|I) v^\sigma_p(h \cdot a)\right)$ where $\pi^\sigma_{-p}(h|I) = \frac{\pi^\sigma_{-p}(h)}{\pi^\sigma_{-p}(I)}$.

[2] Deep RL has also been applied successfully to some partially observed games such as Doom [17], as long as the hidden information is not too strategically important.

Let σ^t be the strategy profile on iteration t. The *instantaneous regret* $r^t(I,a)$ of player $p = P(I)$ for action a in infoset I on iteration t is how much better off p would have been for choosing action a with 100% probability (and playing according to σ^t thereafter) rather than playing according to σ^t in I, weighed by the probability that p would have reached I if he tried to do so that iteration. Formally,

$$r^t(I,a) = \pi^{\sigma^t}_{-p}(I)\big(v^{\sigma^t}(I,a) - v^{\sigma^t}(I)\big) \tag{1}$$

Furthermore, the *regret* on iteration T is $R^T(I,a) = \sum_{t=1}^{T} r^t(I,a)$. Additionally, $R^T_+(I,a) = \max\{R^T(I,a), 0\}$ and $R^T(I) = \max_a\{R^T(I,a)\}$. Regret for p in the entire game is $R^T_p = \max_{\sigma'_p} \sum_{t=1}^{T} \big(u_p(\sigma'_p, \sigma^t_{-p}) - u_p(\sigma^t_p, \sigma^t_{-p})\big)$.

CFR determines an iteration's strategy by applying any of several *regret minimization* algorithms to each infoset [19, 8]. Typically, *regret matching* (RM) is used as the regret minimization algorithm within CFR due to RM's simplicity and lack of parameters [11].

In RM, a player picks a distribution over actions in an infoset in proportion to the positive regret on those actions. Formally, on each iteration $t+1$, p selects actions $a \in A(I)$ according to probabilities

$$\sigma^{t+1}(I,a) = \frac{R^t_+(I,a)}{\sum_{a' \in A(I)} R^t_+(I,a')} \tag{2}$$

If $\sum_{a' \in A(I)} R^t_+(I,a') = 0$ then any arbitrary strategy may be chosen. Typically each action is assigned equal probability, but in this paper we choose the action with highest regret with probability 1, which empirically helps RM better cope with approximation error.

If a player plays according to regret matching in infoset I on every iteration, then on iteration T, $R^T(I) \leq \Delta\sqrt{|A(I)|}\sqrt{T}$ [7]. If a player plays according to CFR on every iteration, then $R^T_p \leq \sum_{I \in \mathcal{I}_p} R^T(I)$. So, as $T \to \infty$, $\frac{R^T_p}{T} \to 0$.

The average strategy $\bar{\sigma}^T_p(I)$ for an infoset I on iteration T is $\bar{\sigma}^T_p(I) = \frac{\sum_{t=1}^{T}\left(\pi^{\sigma^t}_p(I)\sigma^t_p(I)\right)}{\sum_{t=1}^{T}\pi^{\sigma^t}_p(I)}$.

In two-player zero-sum games, if both players' average regret satisfies $\frac{R^T_p}{T} \leq \epsilon$, then their average strategies $\langle \bar{\sigma}^T_1, \bar{\sigma}^T_2 \rangle$ form a 2ϵ-Nash equilibrium [27]. Thus, CFR constitutes an anytime algorithm for finding an ϵ-Nash equilibrium in two-player zero-sum games.

Although CFR theory calls for both players to simultaneously update their regrets on each iteration, in practice far better performance is achieved by alternating which player updates their regrets on each iteration. However, no converge proof is known for such variants [9].

2.2 Linear CFR

There exist a number of variants of CFR that achieve much faster performance. Until recently, CFR+ was the fastest and most popular variant [26]. However, CFR+ does not handle variance well and is therefore difficult to use with sampling and function approximation [23]. In this paper we use *Linear CFR (LCFR)*, a new variant of CFR that is faster than CFR+ in certain settings (particularly in settings with wide distributions in payoffs) even without sampling, and which tolerates error far better than CFR+ [5]. LCFR is identical to CFR, except iteration t is weighed by t. Formally, regret is defined as

$$R^T(I,a) = \sum_{t=1}^{T} \big(t r^t(I,a)\big) \tag{3}$$

and the average strategy is defined as

$$\bar{\sigma}^T_p(I) = \frac{\sum_{t=1}^{T}\left(t\pi^{\sigma^t}_p(I)\sigma^t_p(I)\right)}{\sum_{t=1}^{T}\left(t\pi^{\sigma^t}_p(I)\right)} \tag{4}$$

We will use these definitions for regret and average strategy throughout the paper. Alternating updates with LCFR is in practice two orders of magnitude faster than vanilla CFR in large benchmark games.

3 Related Work

CFR is not the only iterative algorithm capable of solving large imperfect-information games. In addition to first-order methods, there exist algorithms such as Fictitious Play that converge to an equilibrium solution. Neural Fictitious Self Play (NFSP) [12] previously combined deep learning function approximation with Fictitious Play to produce an AI for limit Texas hold'em, a large imperfect-information game. However, Fictitious Play has weaker theoretical convergence guarantees than CFR, and in practice converges far slower. NFSP has not been shown to be competitive with CFR-based approaches. First-order methods converge to a Nash equilibrium in $O(1/T)$ [13, 16], which is far better than CFR's theoretical bound of $O(1/\sqrt{T})$. However, in practice the fastest variants of CFR are substantially faster than the best first-order methods. Moreover, CFR is more robust to error and therefore likely to do better when combined with function approximation.

Past work has investigated using deep learning to estimate values at the depth limit of a subgame in imperfect-information games [21]. However, tabular CFR was used within the subgames themselves.

Prior work has combined linear function approximation with CFR [28] in an algorithm called *Regression CFR (RCFR)*. This algorithm defines a number of features of the infosets in a game and calculates weights to approximate the regrets that a tabular CFR implementation would produce. However, RCFR uses full traversals of the game tree (which is infeasible in large games), a set of pre-defined features, and has only been tested on toy games. It is therefore best viewed as a proof of concept that function approximation can be applied to CFR.

Concurrent work has also investigated a similar combination of deep learning with CFR, in an algorithm referred to as Double Neural CFR [1]. However, the authors consider only small games (thousands of nodes) that can be modeled exactly by a neural network, whereas we are interested in using function approximation to improve the sample complexity of large games (billions to trillions of nodes) by generalizing across infosets using a model that has much fewer parameters than there are infosets. There are important differences between our approaches in how training data is collected and how the behavior of CFR is approximated.

4 Description of the Deep Counterfactual Regret Minimization Algorithm

In this section we describe a way to approximate Linear CFR using deep learning function approximation. The goal of Deep CFR is to approximate the behavior of Linear CFR while avoiding full traversals of the game tree. To begin, we define the *instantaneous advantage* $d^t(I, a)$ of an action as $d^t(I, a) = v^{\sigma^t}(I, a) - v^{\sigma^t}(I)$ and the *advantage* $D^T(I, a)$ as regret (as defined in Equation 3) divided by the *total linear reach* $\sum_{t=1}^{T} \left(t\pi_{-p}^{\sigma^t}(I)\right)$ of the infoset. That is,

$$D^T(I, a) = \frac{R^T(I, a)}{\sum_{t=1}^{T} \left(t\pi_{-p}^{\sigma^t}(I)\right)} \tag{5}$$

Since total linear reach is identical for all actions in an infoset, the formula for RM given in equation (2) can be restated as

$$\sigma^{t+1}(I, a) = \frac{D^t(I, a)_+}{\sum_{a' \in A(I)} D^t(I, a')_+} \tag{6}$$

At a high level, Deep CFR trains a value network $f : I \rightarrow \mathbf{R}^{|A|}$ on each iteration t defined by parameters θ_p^t. This network takes as input a description of an infoset and outputs an estimate of the advantage for each action in the infoset. This output vector is denoted by $\hat{D}^t(I)$ and the prediction of the advantage for action a specifically is denoted by $\hat{D}^t(I, a)$. Ideally, $\hat{D}^t(I, a) \approx D^t(I, a)$. This allows Deep CFR to approximate the behavior of RM to produce strategy $\sigma^{t+1}(I)$ for infoset I on iteration $t + 1$. In order to conduct alternating updates in which only one player's strategy changes on each iteration, Deep CFR maintains a separate set of parameters θ_p^t for each player p. We choose to predict advantages rather than regrets to allow the network to better generalize among strategically similar situations that differ only in how often they are reached during play.

The training data for the network is a set of sampled infoset advantages from iterations 1 through t stored in a buffer B_p^v for player p. We choose our mechanism for collecting samples based on two desirable goals. First, the number of samples for infoset I in B_p^v should, in expectation, be

proportional to the total linear reach $\sum_{t=1}^{T} \left(t\pi_{-p}^{\sigma^t}(I)\right)$ of the infoset. This focuses the network on infosets that have relatively larger regret and are therefore more "important". Second, the samples for infoset I action a should, in expectation, be equal to the true advantage $D^t(I, a)$ of the action.

To achieve these goals, Deep CFR conducts a constant number K of partial traversals of the game tree on each iteration, with the path of the traversal determined according to *external sampling*. In external sampling, the traversal explores all actions in the traversing player's infosets while sampling a single action in opponent infosets and chance nodes. When a terminal node is reached, the value is passed back up. In chance and opponent infosets, the value of the sampled action is passed back up unaltered. In traverser infosets, the value passed back up is the weighted average of all action values, where action a's weight is $\sigma^t(I, a)$. This produces samples of this iteration's contribution to the advantages for the actions in various infosets I. These samples are added to the buffer (using reservoir sampling if capacity is exceeded), and a new network is trained to determine parameters θ^t by minimizing MSE between the predicted advantage $\hat{D}^t(I, a)$ and the samples $\tilde{D}^t(I, a)$ drawn from the buffer. Critically, because advantages are the weighted average over all previous iterations, once a sample is added to the buffer it is never removed (except through reservoir sampling), even when the next CFR iteration begins. However, in order to mimic the linear weighting component of Linear CFR, samples from iteration t are weighed by $\lfloor (t + 1)/2 \rfloor$ rather than all samples in the buffer being weighed equally. While we use MSE as the loss, any Bregman divergence loss is acceptable.

While almost any sampling scheme is acceptable so long as the samples are weighed properly, external sampling has the convenient property that it achieves both of our desired goals by assigning all samples in an iteration equal weight. Additionally, exploring all of a traverser's actions helps reduce variance. However, external sampling may be impractical in games with extremely large branching factors, so a different sampling scheme may be desired in those cases.

In addition to the value network, a separate policy network $g : I \to \mathbf{R}^{|A|}$ approximates the average strategy at the end of the run, because it is the *average strategy played over all iterations* that converges to a Nash equilibrium. To do this, we maintain a separate buffer B_p^s of sampled infoset probability vectors for each player p. Whenever an infoset I belonging to player p is traversed during a player $1 - p$ traversal of the game tree via external-sampling, the infoset probability vector $\sigma^t(I)$ is added to B_p^s and assigned weight t.

Theorem 1 states that if the buffers used in Deep CFR are infinitely large, we conduct an infinite number of traversals to collect data, and our function approximator achieves the minimum possible error on the value network and the minimum possible error on the policy network, then Deep CFR perfectly mimcs Linear CFR.

Theorem 1. *Assume Deep CFR conducts K traversals on each iteration of CFR, the value network and policy network buffers are infinitely large, and the function approximator achieves the minimum possible error on the value network and the policy network. Then as $K \to \infty$, Deep CFR's convergence bound approaches the convergence bound of Linear CFR.*

Algorithm 1 Deep Counterfactual Regret Minimization

$B_0^r = \emptyset, B_1^r = \emptyset, B^s = \emptyset$
for $p = 1..2$ **do**
  Initialize θ_p^0 so that $f(I, a|\theta_p^0) = 0$ for all I and all a
for $t = 1..N_{iter}$ **do**
  $p \leftarrow t \,\%\, 2$
  for $n = 1..N_{traversal}$ **do**
    CollectSamples($\emptyset, p, \theta_0^{t-1}, \theta_1^{t-1}, B_p^r, B^s$)
  $\theta_p^t \leftarrow$ TrainNetwork($B_p^r, 0$) ▷ Retrain the value network incorporating newly collected data
  $\theta_{1-p}^t \leftarrow \theta_{1-p}^{t-1}$ ▷ We update only one player's value network per iteration
$\theta^{(s)} \leftarrow$ TrainNetwork($B^s, 1$) ▷ Train the final average strategy
return $\theta^{(s)}$

Algorithm 2 Sample Collection Traversal

function COLLECTSAMPLES($h, p, \theta_0, \theta_1, B_p^r, B^s$)
  if h is terminal **then**
    return $u_p(h)$ ▷ Return the traverser's payoff
  else if $P(h) = p$ **then** ▷ If it's the traverser's turn to act
    $\sigma(I) \leftarrow$ CALCULATE-STRATEGY($I(h), \theta_p$) ▷ Compute infoset action probabilities
    $v \leftarrow 0$
    for $a \in A(h)$ **do**
      $v(a) \leftarrow$ COLLECTSAMPLES($h \cdot a, p, \theta_0, \theta_1, B_p^r, B^s$) ▷ Traverse each action
      $v \leftarrow v + \sigma(I, a) \cdot v(a)$ ▷ Update the expected value
    for $a \in A(h)$ **do**
      $\tilde{d}(I, a) \leftarrow v(a) - v$
    Add $\{(I, \tilde{d}(I), t)\}$ to B_p^r ▷ Add vector of action advantages to buffer
  else if $P(h) = 1 - p$ **then** ▷ If it's the opponent's turn to act
    $\sigma(I) \leftarrow$ CALCULATE-STRATEGY($I(h), \theta_{1-p}$) ▷ Compute infoset action probabilities
    Add $\{(I, \sigma(I), t)\}$ to B^s ▷ Add vector of action probabilities to buffer
    $a \sim \sigma(I)$ ▷ Sample an action from the probability distribution
    return COLLECTSAMPLES($h \cdot a, p, \theta_0, \theta_1, B_p^r, B^s$)
  else ▷ h is a chance node
    $a \sim \sigma(h)$ ▷ Sample a chance outcome
    return COLLECTSAMPLES($h \cdot a, p, \theta_0, \theta_1, B_p^r, B^s$)

Algorithm 3 Infoset Strategy Computation

function CALCULATE-STRATEGY(I, θ_p) ▷ Calculates strategy based on predicted advantages
  sum $\leftarrow 0$
  $\hat{D}(I) \leftarrow f(I|\theta_p)$ ▷ Estimate vector of advantages
  for $a \in A(I)$ **do**
    sum $\leftarrow$ *sum* $+ \max\{0, \hat{D}(I, a)\}$
  if *sum* > 0 **then** ▷ Apply Regret Matching
    for $a \in A(I)$ **do**
      $\sigma(I, a) \leftarrow \frac{\max\{0, \hat{D}(I,a)\}}{sum}$
  else ▷ Choose the highest-advantage action
    for $a \in A(I)$ **do**
      $\sigma(I, a) \leftarrow 0$
    $\sigma(I, \text{argmax}_a\{\hat{D}(I, a)\}) = 1$
  return $\sigma(I_i)$

Algorithm 4 Network Training

function TRAINNETWORK(B, S)
  Initialize θ randomly.
  for $b = 1..N_{train}$ **do**
    for $i = 1..N_{batch}$ **do**
      $(I_i, y_i, t_i) \sim B$ ▷ sample an infoset, action pair from the buffer
      $\hat{z} \leftarrow f(I_i|\theta)$ ▷ predict regret or strategy vector
      if S **then**
        for $a \in A$ **do**
          $\hat{y}_{i,a} \leftarrow \frac{e^{\hat{z}_a}}{\sum_{a'} e^{\hat{z}_{a'}}}$ ▷ apply softmax if computing strategy vector
      else
        $\hat{y}_i \leftarrow \hat{z}$
    $\mathcal{L} \leftarrow \sum_0^{N_{batch}} t_i (y_i - \hat{y}_i)^2$
    $\theta \leftarrow \text{StepAdam}(\theta, \nabla_\theta \mathcal{L})$
  return θ

5 Experimental Setup

We measure the performance of Deep CFR (Algorithm 1) in approximating an equilibrium in heads-up flop hold'em poker (FHP). FHP is a large game with over 10^{11} nodes and 10^8 infosets. In contrast, the network we use has 61,604 parameters. FHP is similar to heads-up no-limit Texas hold'em (HUNL) poker, but has only two betting rounds rather than four, and all bets and raises must be equal to the size of the pot. The rules for FHP are given in Appendix A.

We also measure performance relative to domain-specific abstraction techniques in the benchmark domain of HUNL poker, which has about 10^{161} infosets. A standard approach to developing an AI for such large imperfect-information games is to first limit the action space to a small discrete number and solve the coarsened version of the game. This solution is referred to as the *blueprint* strategy [4, 6]. Then, real-time solving is applied on top of the blueprint strategy to calculate responses to actions not included in the blueprint. Real-time solving is beyond the scope of this paper, so we simply compare the performance of a blueprint strategy computed with traditional information-abstraction techniques [14, 10, 3] to one computed with Deep CFR. The blueprint version of HUNL we test on requires all bets and raises to be equal to the size of the pot. This blueprint game has over 10^{15} nodes and over 10^{12} infosets.

5.1 Network Architecture

We use the neural network architecture shown in Figure 5.1 for both the network that computes infoset and action values for each player and the network that approximates the average strategy at the end of MC-CFR. This network has a depth of 7 layers and 61,604 parameters. Infosets consist of sets of cards and bet history. The cards are represented as the sum of three embeddings: a rank embedding (1-13), a suit embedding (1-4), and a card embedding (1-52). These embeddings are summed for each set of permutation invariant cards (hole, flop, turn, river), and these are concatenated. In each round of betting we specify a maximum number of sequential actions, leading to $N_{rounds}N_{seq-actions}$ total unique betting positions. Each betting position is encoded by three numbers: the first number is binary and denotes if a bet was made; the second denotes the size of the bet; the third denotes if this is the current betting position.

The neural network model begins with separate branches for the cards and bets, with three and two layers respectively. Features from the two branches are combined and three additional fully connected layers are applied. Each fully-connected layer consists of $x_{i+1} = \text{ReLU}(Ax[+x])$. The optional skip connection $[+x]$ is applied only on layers that have equal input and output dimension. Normalization (to zero mean and unit variance) is applied to the last-layer features of each position. The network architecture was not highly tuned, but normalization and skip connections were used because they were found to be important to encourage fast convergence when running preliminary experiments on pre-computed equilibrium strategies in FHP.

There are at most a few thousand infosets on the first betting round, so we simply use a table rather than function approximation to determine the regrets and average strategies for the first betting round.

5.2 Model training

The value model is trained from scratch each CFR iteration, starting from a random initialization. We perform 4,000 mini-batch SGD iterations using a batch size of 10,000 and perform parameter updates using the Adam optimizer [15] with a learning rate of 0.001, with gradient norm clipping to 1[3].

We allocate a maximum size of 40 million infosets to each player's value buffer B_p^v and the strategy buffer B^s. Once a buffer is full it is updated via reservoir sampling, maintaining a uniform distribution over infoset values from all prior iterations. We record the iteration at which each sample was collected to weight the training loss for linear CFR.

6 Experimental Results

Figure 6 compares the performance of Deep CFR to variously-sized domain-specific abstractions. The abstractions are solved using external-sampling Linear Monte Carlo CFR [18, 5], which is the state

[3]For HUNL we use 32,000 SGD iterations and a batch size of 20,000.

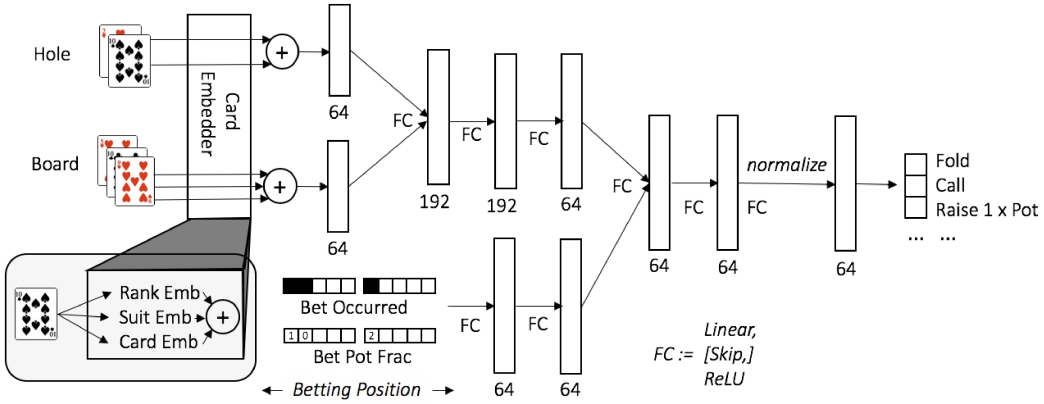

Figure 1: The neural network architecture used for Deep CFR. The network takes an infoset (observed cards and bet history) and outputs values (advantages or probability logits) for each possible action.

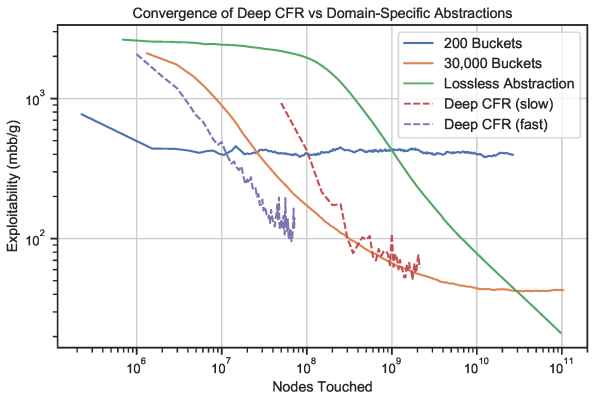

Figure 2: Comparison of Deep CFR with domain-specific abstractions in FHP. Coarser abstractions converge faster but are more exploitable. Deep CFR converges with 1-2 orders of magnitude fewer samples than a lossless abstraction, and performs competitively with a 30,000 bucket abstraction. *fast* and *slow* Deep CFR curves use 30,000 and 2,000,000 CFR traversals per step, respectively.

of the art algorithm in this setting. The 30,000 bucket run means that the 25,989,600 different poker hands on the second betting round were clustered into 30,000 buckets, where the hands in the same bucket are treated identically. This bucketing is done using K-means clustering on domain-specific features. Lossless abstraction only buckets together poker hands that are strategically isomorphic (e.g., flushes that differ only by suit), so a solution to this abstraction maps to a solution in the full game without error. This results in 1,286,792 buckets.

The figure shows that Deep CFR's performance, measured by number of nodes visited, is comparable to the abstraction in which the 26 million different poker hands on the flop are clustered into 30,000 buckets. Although performance is comparable in terms of nodes touched, neural network inference and training requires considerable overhead that tabular CFR avoids. However, Deep CFR does not require advanced domain knowledge. We plot deep CFR performance for both 30,000 (*fast*) and 2,000,000 (*fast*) CFR traversals per step. Using a small number of CFR traversals per step is more sample efficient but requires more CFR steps and thus greater neural network training time.

Figure 6 shows the performance of Deep CFR for different numbers sample-collecting traversals and different numbers of neural network training SGD steps, per CFR iteration. Diminishing returns are observed in both cases. We therefore observe there is a tradeoff between doing a small number of highly accurate CFR steps (using a lot of data and a large number of SGD steps) versus doing a larger

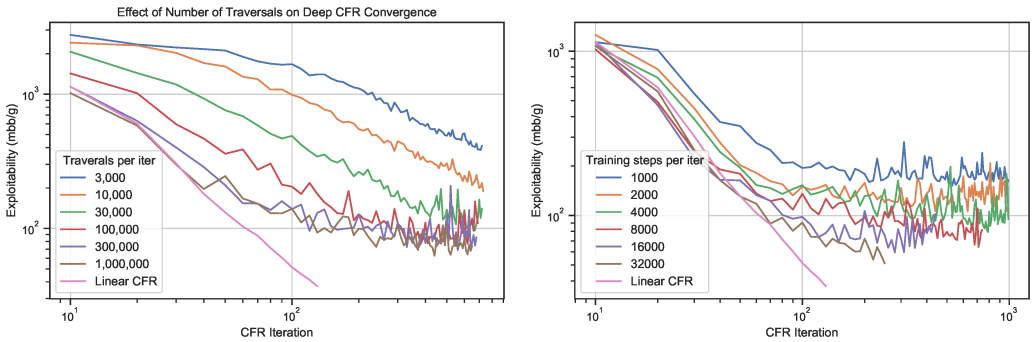

Figure 3: **Left:** FHP convergence for different numbers of training data collection traversals per simulated LCFR iteration. **Right:** FHP convergence using different numbers of minibatch SGD updates per simulated LCFR iteration to train the advantage model. We also compare to tabular unsampled Linear CFR.

number of CFR steps that are more inaccurate. The optimal choice ultimately depends on the desired accuracy and the cost of training the network relative to conducting traversals of the game tree.

Finally, we measure head-to-head performance in a variant of HUNL in which all bets and raises must equal the size of the pot. We compare Deep CFR to a strategy in which Monte Carlo CFR was run on an abstraction in which on each betting round all the different poker hands were bucketed into 30,000 buckets. Our Deep CFR agent beats the abstraction-based strategy by 89 ± 11 mbb/g. For comparison, Tartanian7, the winning agent of the 2014 Annual Computer Poker Competition also used 30,000 buckets per round and defeated all other agents with statistical significance, with the nearest competitor losing by 20 ± 16 mbb/g [3].

7 Conclusions

We introduced a deep neural network implementation of CFR that is theoretically principled and achieves strong performance relative to domain-specific abstraction techniques while not relying on advanced domain knowledge in the benchmark game of HUNL poker. This is the first successful application of deep learning to CFR in a large game.

References

[1] Anonymous Authors. Double neural counterfactual regret minimization. *https://openreview.net/pdf?id=Bkeuz20cYm*, 2018.

[2] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit hold'em poker is solved. *Science*, 347(6218):145–149, January 2015.

[3] Noam Brown, Sam Ganzfried, and Tuomas Sandholm. Hierarchical abstraction, distributed equilibrium computation, and post-processing, with application to a champion no-limit texas hold'em agent. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 7–15. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[4] Noam Brown and Tuomas Sandholm. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, page eaao1733, 2017.

[5] Noam Brown and Tuomas Sandholm. Solving imperfect-information games via discounted regret minimization. *arXiv preprint arXiv:1809.04040*, 2018.

[6] Noam Brown, Tuomas Sandholm, and Brandon Amos. Depth-limited solving for imperfect-information games. In *Advances in Neural Information Processing Systems*, 2018.

[7] Nicolo Cesa-Bianchi and Gabor Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006.

[8] Kamalika Chaudhuri, Yoav Freund, and Daniel J Hsu. A parameter-free hedging algorithm. In *Advances in neural information processing systems*, pages 297–305, 2009.

[9] Gabriele Farina, Christian Kroer, and Tuomas Sandholm. Online convex optimization for sequential decision processes and extensive-form games. *arXiv preprint arXiv:1809.03075*, 2018.

[10] Sam Ganzfried and Tuomas Sandholm. Potential-aware imperfect-recall abstraction with earth mover's distance in imperfect-information games. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2014.

[11] Sergiu Hart and Andreu Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68:1127–1150, 2000.

[12] Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016.

[13] Samid Hoda, Andrew Gilpin, Javier Peña, and Tuomas Sandholm. Smoothing techniques for computing Nash equilibria of sequential games. *Mathematics of Operations Research*, 35(2):494–512, 2010. Conference version appeared in WINE-07.

[14] Michael Johanson, Neil Burch, Richard Valenzano, and Michael Bowling. Evaluating state-space abstractions in extensive-form games. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multiagent Systems*, pages 271–278. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] Christian Kroer, Kevin Waugh, Fatma Kılınç-Karzan, and Tuomas Sandholm. Faster algorithms for extensive-form game solving via improved smoothing functions. *Mathematical Programming*, pages 1–33.

[17] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *AAAI*, pages 2140–2146, 2017.

[18] Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. Monte Carlo sampling for regret minimization in extensive games. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1078–1086, 2009.

[19] Nick Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[21] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 2017.

[22] John Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36:48–49, 1950.

[23] Martin Schmid, Neil Burch, Marc Lanctot, Matej Moravcik, Rudolf Kadlec, and Michael Bowling. Variance reduction in monte carlo counterfactual regret minimization (vr-mccfr) for extensive form games using baselines. *arXiv preprint arXiv:1809.03057*, 2018.

[24] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[25] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[26] Oskari Tammelin, Neil Burch, Michael Johanson, and Michael Bowling. Solving heads-up limit texas hold'em. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 645–652, 2015.

[27] Kevin Waugh. Abstraction in large extensive games. Master's thesis, University of Alberta, 2009.

[28] Kevin Waugh, Dustin Morrill, Drew Bagnell, and Michael Bowling. Solving games with functional regret estimation. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2015.

[29] Martin Zinkevich, Michael Johanson, Michael H Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1729–1736, 2007.

A Flop Hold'em Poker Rules

In the version of FHP we use in this paper, there are two players and the position of the two players alternate after each hand. On each betting round, each player can choose to either fold, call, or raise. Folding results in the player losing and the money in the pot being awarded to the other player. Calling means the player places a number of chips in the pot equal to the opponent's share. Raising means that player adds more chips to the pot than the opponent's share. A round ends when a player calls (if both players have acted). If a player raises, they first put into the pot the number of chips the other player has contributed so far, and then an additional number of chips equal to the size of the new pot (a pot-sized raise). There cannot be more than one raise in the first betting round or more than two raises in the second betting round, so there is a limited number of actions in the game. Additionally, calling is not allowed for the first action in the game.

At the start of each hand of FHP, both players are dealt two private cards from a standard 52-card deck. P_1 must place \$50 in the pot and P_2 must place \$100 in the pot. A round of betting then occurs starting with P_1. When the round ends, three *community* cards are dealt face up that both players can ultimately use in their final hands. Another round of betting occurs, starting with P_2 this time. At the end of the betting round, unless a player has folded, the player with the best five-card poker hand constructed from their two private cards and the five community cards wins the pot. In the case of a tie, the pot is split evenly.

B Proof of Theorem 1

Proof. Consider an information set I in which player p acts. Let $\tilde{v}^{\sigma^t}(I,a)$ be the expected value return by the external sampling iteration for infoset I action a and let $\tilde{v}^{\sigma^t}(I,a)$ be the value for infoset I. From Lemma 1 in [18], we know that $E[\tilde{v}^{\sigma^t}(I)] = v^{\sigma^t}(I)$ and $E[\tilde{v}^{\sigma^t}(I,a)] = v^{\sigma^t}(I,a)$. Since $K \to \infty$, so the average of all samples $\tilde{d}^{\sigma^t}(I,a) = \tilde{v}^{\sigma^t}(I,a) - \tilde{v}^{\sigma^t}(I)$ in the buffer approaches $E[\tilde{v}^{\sigma^t}(I,a) - \tilde{v}^{\sigma^t}(I)] = v^{\sigma^t}(I,a) - v^{\sigma^t}(I) = d^{\sigma^t}(I,a)$.

Let n_t be the number of samples of I on iteration t. The probability that I is sampled during a traversal on iteration t is $\pi^{\sigma^t}_{-p}(I)$. Since there are K traversals on each iteration, so $E[n_t] = K\pi^{\sigma^t}_{-p}(I)$ and $E[\frac{tn_t}{\sum_{t'=1}^{T}(tn_{t'})}] = \frac{t\pi^{\sigma^t}_{-p}(I)}{\sum_{t'=1}^{T}\left(t\pi^{\sigma^{t'}}_{-p}(I)\right)}$. Since $K \to \infty$, so $\frac{tn_t}{\sum_{t'=1}^{T}(tn_{t'})} \to E[\frac{tn_t}{\sum_{t'=1}^{T}(tn_{t'})}] = \frac{t\pi^{\sigma^t}_{-p}(I)}{\sum_{t'=1}^{T}\left(t\pi^{\sigma^{t'}}_{-p}(I)\right)}$.

Since samples from iteration t are assigned weight t and since the value network achieves minimum error, so $\hat{D}^T(I,a) = \frac{\sum_{t=1}^{T}\left(t\pi^{\sigma^t}_{-p}(I)d^{\sigma^t}(I,a)\right)}{\sum_{t=1}^{T}\left(t\pi^{\sigma^t}_{-p}(I)\right)} = D^T(I,a)$.

Similarly, since samples from iteration t are assigned weight t and since the policy network achieves minimum error, so $\hat{\sigma}^T(I,a) = \frac{\sum_{t=1}^{T}\left(t\pi^{\sigma^t}_{-p}(I)\sigma^t(I,a)\right)}{\sum_{t=1}^{T}\left(t\pi^{\sigma^t}_{-p}(I)\right)} = \bar{\sigma}^T(I,a)$.

□